GPU SMITH
COMPUTE ENGINEERING

GPU SMITH / RESEARCH & PRACTICAL GUIDANCE

Slurm GPU Isolation and Cleanup: A Launch-Path Matrix

> Plan and build private AI compute with GPU Smith. We connect workload requirements with GPU hardware, networking, deployment and operating decisions so your infrastructure fits the work it must perform.

Published by GPU Smith · 2026-11-25 · slurm gpu isolation · slurm constraindevices · slurm cgroup gpu · cuda_visible_devices · gres gpu configuration · pam_slurm_adopt · slurm epilog gpu cleanup · cgroup v2 ebpf · slurm srun sbatch gpu

Original article: https://gpusmith.com/articles/en/slurm-gpu-isolation-cleanup-acceptance-matrix



In this article

1. Executive Summary
2. Introduction and Background
3. Key Changes in the Isolation Model: Five Layers That Must Be Tested Separately
4. The Launch-Path Acceptance Matrix
5. Termination and Cleanup Cases
6. Implementation Considerations and Process Changes
7. Data Analysis and Evidence
8. Implications and Future Directions
9. Frequently Asked Questions (FAQs)
10. Conclusion

Executive Summary

A Slurm GPU allocation is four different things that are easy to conflate: a **count** held by the controller, an **environment mask** exported to the job, an **enforced device-access rule** in the kernel, and a **release** of the device when the job ends. Slurm documents that no generic resources (GRES) are enabled by default, and that without a `File=` entry in `gres.conf` it tracks only a count of allocated resources, not individual device files (slurm.schedmd.com) (manpages.ubuntu.com). Enforcement of the file allocation relies on Linux control groups (cgroups) and the `task/cgroup` plugin (manpages.ubuntu.com), and the `ConstrainDevices` option in `cgroup.conf` is documented as constraining a job's allowed devices based on its GRES, with a default of "no" (man.archlinux.org). A job that sees a correct `CUDA_VISIBLE_DEVICES` value has therefore shown only the mask.

This report proposes an **allocation-to-device-access-to-cleanup acceptance contract** that is exercised per launch path: batch (`sbatch`), `srun` steps, SSH sessions adopted by `pam_slurm_adopt`, and the site's real container or serving launcher. NVIDIA's AI Cluster Runtime (AICR) validator already implements a Slinky-specific one-GPU versus zero-GPU `srun` probe, and it states that the cgroup device controller "leaves the nodes visible and refuses" the open call (docs.nvidia.com). The contribution here is a portable matrix that adds SSH, the real launcher and termination cases, not a replacement for that probe.

Three version-sensitive facts shape any test script. First, cgroup v1 support is marked deprecated from Slurm 25.05 onward (raw.githubusercontent.com), and on cgroup v2 the devices controller is replaced by an eBPF (extended Berkeley Packet Filter) program (raw.githubusercontent.com). Second, the Slurm 26.05 release notes record that cgroup/v2 directory structures are keyed off SLUID (a job identifier) and not the JobId (raw.githubusercontent.com), with `CgroupJobIdPaths` in `cgroup.conf` as the switch back to numeric job-ID names. Third, `pam_slurm_adopt` defaults can let a login through without adoption, so the module's failure path needs its own controlled test (slurm.schedmd.com).

Release is measured, not assumed. The default cancel sequence is SIGCONT, then SIGTERM, a wait of KillWait, then SIGKILL (manpages.debian.org), and a step that outlives SIGKILL by the unkillable-step timeout (60 seconds or five times MessageTimeout, whichever is greater) drains the node (man.archlinux.org). Cleanup latency is defined as the time the last job-owned GPU context is confirmed released minus the time

termination was requested, with grace periods and sampling resolution reported separately. Coverage is reported as executed applicable cases over declared applicable cases, alongside failed and unknown counts, and it is explicitly not a security score. No isolation test, fault injection or benchmark was performed for this report; every finding below is a documented property or an acceptance procedure, not a measured result.

Introduction and Background

Private Slurm operators ask one question in several forms: does a job see only the GPUs it was given, and are those GPUs free for the next job after cancellation, timeout or a launcher failure? The search phrases around "slurm gpu isolation" and "slurm constrain devices" mix a decade of mailing-list posts with current documentation. This article therefore treats current SchedMD documentation as the reference and leaves site behavior to measurement on the reader's own canary node.

Scope. This report narrows to whole NVIDIA GPUs used by ordinary unprivileged Slurm users on a private cluster. It does not cover GPU sharing modes, accounting or chargeback, or advice to terminate processes by user name. Multi-Instance GPU (MIG) and AMD devices are excluded until their device and capability mappings are separately verified; NVIDIA documents MIG capability nodes as a distinct mechanism from the traditional cgroup control of top-level GPU device nodes (docs.nvidia.com). A device-open policy is also not a complete GPU security or confidentiality guarantee: it governs who may open a device node, not what a driver, firmware or administrator can observe.

Trust boundary. The allocation is a promise made by the controller to an unprivileged job. Administrators with root, the Slurm daemons and the prolog and epilog (documented as running outside any job cgroup while the job runs inside one (slurm.schedmd.com)) sit on the other side of that boundary, so prolog-side checks cannot be copied into job-side checks without translation.

Perspective of the publisher. GPU Smith is a private AI infrastructure engineering consultancy. It holds no test results for the procedures below. The matrix is a set of reproducible assumptions, and every site-specific behavior (PAM stack, launcher, kernel, driver) must be established on the reader's own canary node. Statements that are inferences, not documented facts, are labeled as such.

Key Changes in the Isolation Model: Five Layers That Must Be Tested Separately

The central design rule is that each layer below can pass while another fails. A test that observes only one layer cannot support a statement about the others.

Layer 1: Resource counting

Counting decides whether the scheduler places a job on a node with a free GPU. Slurm's GRES guide states that by default no GRES are enabled and that jobs receive no generic resources unless they request them (slurm.schedmd.com). The `gres.conf` documentation adds that without `File=`, Slurm only counts allocated resources instead of tracking the state of each device file. A cluster can therefore schedule correctly and enforce nothing. The `slurm.conf` `GresTypes` default is likewise to manage no generic resources (manpages.ubuntu.com).

Layer 2: Environment masking

The mask is the variable the application reads. CUDA documents that when `CUDA_VISIBLE_DEVICES` is unset all GPU devices are visible, that an empty string means no devices are visible, and that an invalid index truncates the list so only earlier entries stay visible (docs.nvidia.com).

Slurm sets the variable only when tasks are launched on a compute node, so `salloc` and the prolog do not see the job's value (slurm.schedmd.com). Operators describe the variable as set automatically to the job's allocated GPUs (hpc.ncsu.edu), and a SchedMD support engineer warns that relying only on environment variables is not safe for accessing devices (support.schedmd.com). The prolog and epilog can see different device mappings from the job: the Prolog Epilog guide explains that they run outside the job's cgroup and may thus see a different set of device files (slurm.schedmd.com). Slurm also warns that mappings between device-file numbers and detected indices are system dependent and can vary between boots, and that an after-bootup check is required (slurm.schedmd.com). NVIDIA's own guidance is to identify GPUs by UUID or PCI bus ID for consistency, since enumeration order is not guaranteed across reboots (docs.nvidia.com). The `Flags=env_uuid` setting exports UUIDs instead of numeric indices (man.archlinux.org), which makes mask evidence comparable to `nvidia-smi` evidence.

Two traps follow for test authors. The CUDA default device order is a "fastest first" heuristic rather than PCI order (docs.nvidia.com), so a bare index is not stable evidence unless `CUDA_DEVICE_ORDER=PCI_BUS_ID` is set (docs.nvidia.com), a requirement Slurm's NVML numbering note also raises. And `SLURM_JOB_GPUS` carries global GPU IDs that are not relative to any device cgroup, even when devices are constrained (slurm.schedmd.com). Index zero inside a job is not evidence of physical GPU zero.

Layer 3: Enforced device access

Enforcement is the kernel refusing an `open()` call. The `gres.conf` guide says `File=` is generally required if enforcement of GRES allocations is to be supported (slurm.schedmd.com), and the cgroups overview lists confining jobs, steps and tasks to their allocated GRES, including GPUs, as a `task/cgroup` function (slurm.schedmd.com). On cgroup v1 the kernel's devices controller is a whitelist edited through `devices.allow` and `devices.deny` (docs.kernel.org). On cgroup v2 there are no interface files: the kernel documents the v2 device controller as implemented on cgroup BPF (docs.kernel.org), and Slurm describes creating a BPF program dynamically in `slurmstepd` and inserting it. Slurm manages only the devices described in `gres.conf` (slurm.schedmd.com), so a device node that `gres.conf` never names is outside the rule the plugin builds. The same eBPF mechanism underlies systemd's `DeviceAllow=` (www.freedesktop.org). The eBPF reference describes these programs as running when a process in the cgroup wants to use a device (docs.ebpf.io), and a return of 0 fails the attempt with an error (docs.ebpf.io).

NVIDIA's device model explains why file listings mislead. Each physical GPU has its own `nvidiaN` node, and a few system-wide nodes such as `nvidiactl` and `nvidia-uvm` accompany them (docs.nvidia.com). The driver FAQ notes that device files are created with default ownership and mode unless module parameters override them (download.nvidia.com). The AICR validator makes the same point operationally: its probe never lists `/dev/nvidia*` to infer isolation, because the cgroup device controller leaves the nodes visible and refuses the open.

The path of the cgroup also changed. The current `cgroup.conf` documents that SLUID now names job-level cgroup/v2 directories, and that `CgroupJobIdPaths=yes` restores numeric job-ID naming for external tooling (slurm.schedmd.com). Probes should therefore obtain the path from the process itself. The `/proc/PID/cgroup` file lists one entry per hierarchy (man7.org), and for the v2 hierarchy the hierarchy ID field is 0 (man7.org).

Layer 4: Process membership

A GPU context belongs to a process, and a process belongs to exactly one cgroup per hierarchy (man7.org). A forked child inherits its parent's cgroup memberships (man7.org), and the kernel changes membership only when a PID is written to another cgroup's `cgroup.procs` (docs.kernel.org). Slurm's `slurm.conf` says `proctrack/cgroup` is highly recommended to track all processes reliably (manpages.ubuntu.com) and is the default on cgroup-capable systems (man.archlinux.org). A cloud operator guide says it keeps all processes created by the job in a cgroup (awslabs.github.io), but that a process started through tmux outside Slurm is not cleaned up by cgroups (awslabs.github.io).

Reparenting and membership are separate kernel properties. A subreaper or init adopts orphans (man7.org), `setsid` makes the caller the only member of a new session and process group (man7.org), and the parent-death signal setting is cleared for forked children (man7.org). None of those pages describes a cgroup move. This is an inference from separate pages and not a sourced statement, but it has a testable consequence: seeing PID 1 as the parent of a surviving worker does not show that the worker left the job's cgroup. The test must read `/proc/PID/cgroup` and record ownership, not infer escape from `PPID`.

One documented exception is a genuine way out. Slurm states that processes started through systemd (for example with `systemd-run`) escape the job cgroup, and that Slurm cannot prevent it (slurm.schedmd.com). Whether a given site blocks that route is a site-specific control to test, not an assumption.

Layer 5: Post-job release

Release is the least documented layer. A zombie remains associated with its cgroup until reaped (docs.kernel.org), and a terminated child that has not been waited for is a zombie (man7.org). The adopting init reaps those entries automatically (man7.org), so a zombie is a leftover process-table entry rather than a running GPU context. A process in uninterruptible sleep (state D) is listed separately in the process state codes (man7.org), and SIGKILL cannot be caught, blocked or ignored (man7.org), so a process that survives SIGKILL is, by inference from these two pages, blocked in the kernel and not ignoring the signal. On the driver side, GPU state stays loaded while any client holds the device file open, and unloads once all clients close it unless persistence is in effect (docs.nvidia.com).

For the controller, the Epilog runs on every node of an allocation when the allocation is released, the node stays in a completing state while the epilog runs, and a non-zero epilog exit code drains the node (man.archlinux.org). The kernel's `cgroup.kill` file can signal every process in a cgroup tree and handles concurrent forks (docs.kernel.org); LWN describes writing to it as the immediate death of every process in the group (lwn.net) while noting that kernel threads are not taken down (lwn.net). The `populated` key in `cgroup.events` reports whether any live process remains (docs.kernel.org). Those two files are useful instruments for the test harness, and neither replaces observing the GPU.

The Launch-Path Acceptance Matrix

An **acceptance contract** here means a written list of cases, each with an expected outcome, an evidence artifact and a disposition rule. A case belongs to a route (how the process enters the allocation) and an event (what ends it). *Table 1* below defines the routes and the evidence each must retain; *Table 2* later defines the failure taxonomy.

Route	Entry and allocation	Evidence to retain	Documented behavior that shapes the test
Batch (`sbatch`)	Batch step inside the job allocation with one GPU requested	`nvidia-smi -L` UUID list, `/proc/self/cgroup`, read-only `open()` errno per `/dev/nvidiaN`, a short CUDA kernel run	`CUDA_VISIBLE_DEVICES` is set when tasks launch on the node; a step inherits all of the job's GRES by default (manpages.debian.org)
`srun` step	Step inside an existing allocation, including a step that requests zero GPUs	Same artifacts per step; step-level cgroup path	`srun` inside an `sbatch` or `salloc` allocation creates a job step (computing.ch.cam.ac.uk); setting the GRES count to zero gives a step none (manpages.debian.org)
SSH-adopted session	`ssh` to the allocated node as the job owner, with `pam_slurm_adopt` in the PAM account stack	`/proc/self/cgroup` showing the extern step; `open()` errno; UUID list	Adoption relies on the job container that `PrologFlags=Contain` creates on allocated nodes (manpages.ubuntu.com), and the default failure policy can let the connection through unadopted
Site container or serving launcher	The real launcher (OCI bundle, Enroot with Pyxis, Apptainer, or an inference server) inside a job	Cgroup path of the containerized PID, runtime device list, UUIDs seen inside	Container features are limited by the OCI runtime used (slurm.schedmd.com); Apptainer makes all host devices available by default (apptainer.org)

The matrix is intentionally not exhaustive. It names the four entry points that a private cluster typically exposes, and any additional entry point (an interactive allocation, a web gateway, a workflow engine) should be added as a fifth row before coverage is reported.

Table 1 should be read together with the observation that routes differ in who sets the mask and who enforces access. Batch and `srun` steps are launched by `slurmstepd`, which moves user processes into task cgroups (slurm.schedmd.com). The SSH route enters through `sshd` and PAM, and the container route enters through whichever runtime the site chose. A pass on the first route is not transferable to the others.

Positive control

The positive control proves that enforcement is not simply denying everything. Submit a one-GPU job as an unprivileged canary user, record the UUIDs that `nvidia-smi` reports inside the job, and compare them with the UUID the scheduler assigned (obtained through the stable-identity reconciliation described later). Then initialize CUDA and execute a short kernel on that device. NVIDIA documents the UUID as a globally unique immutable identifier (docs.nvidia.com), and the PCI bus ID is the other stable key. The AICR probe uses a comparable structure: its first job must list exactly one GPU and open exactly one GPU minor device.

Negative controls

Negative controls are run on an authorized canary node and never against another tenant's active device. They are:

- **No-GPU job.** A job or step that requests no GPU must open no GPU device. The AICR probe pins this job to the same Slurm node as the one-GPU job and expects it to open none (docs.nvidia.com).
- **Non-allocated device.** From inside a one-GPU job, attempt a read-only `open()` of a device node that was not allocated and record the errno, as the AICR probe records the errno of a read-only open of each device node (docs.nvidia.com). A denial and a success are different results from a mere listing.
- **Mask change.** Set `CUDA_VISIBLE_DEVICES` to a different index or UUID, then to an empty string, then unset it. CUDA's documented semantics (all visible when unset, none when empty) define what the application sees, but the device-open attempt must still be denied for non-allocated devices. Passing only the CUDA-level result shows the mask worked and says nothing about enforcement.

Observe the device-open outcome and the CUDA outcome as separate columns. `nvidia-smi` success alone also does not prove confinement: NVIDIA's documentation says the tool's text output is not guaranteed to be backwards compatible (docs.nvidia.com), and the NVML process listing returns only compute processes (nvidia.github.io).

Repeating on each launch path

For **batch and `srun`**, run the three negative controls as separate steps inside one allocation: one step with the default GRES, one with zero GPUs, and one with an altered mask. Slurm documents that `--gpu-bind` mask and map IDs are zero-based indexes relative to the job's GRES only when `task/cgroup` and `ConstrainDevices` are in effect, and global IDs otherwise (manpages.debian.org), so a binding test doubles as a probe of the configuration actually in force.

For the **SSH-adopted route**, operator documentation describes `pam_slurm_adopt` as blocking SSH logins to compute nodes where the user has no running job (docs.coreweave.com), with ownership meaning a running job or an allocation (hpc-wiki.itc.rwth-aachen.de), and as tracking an allowed session as a job step (docs.coreweave.com). The session's `/proc/self/cgroup` should name the job's extern step, and its device-open attempts should match the allocation. Current SchedMD documentation does not settle whether an adopted session inherits the device constraint, so the SSH device-open result is the evidence.

For the **site launcher**, run the same controls with the exact image, runtime and flags used in production. Pyxis is described as a SPANK plugin that manages containerized jobs in Slurm (docs.aws.amazon.com), and Stanford's Sherlock documentation says it lets users run containers as native Slurm job steps (www.sherlock.stanford.edu). Slurm's container documentation states that containers are limited by the OCI runtime used, that rootless Podman by default creates its own cgroup namespace and uses the systemd cgroup manager (slurm.schedmd.com), and that the documented configuration keeps it visible to and constrained by Slurm's cgroup limits including devices (slurm.schedmd.com). Enroot describes cgroup inheritance but also little to no isolation (github.com), and Apptainer with `--nv` makes all NVIDIA devices on the host present in the container (apptainer.org). Each of those statements describes a design that relies on an outer control. The test confirms that the outer control is present for this launcher.

Termination and Cleanup Cases

Cleanup cases are crossed with every route in *Table 1*. The events are listed below, and each case records timestamps, cgroup membership and GPU contexts before and after.

- **Normal completion** of the job.
- **`scancel` of the job**, which starts the signal sequence below.
- **Cancellation of a single step**, which leaves the allocation in place.
- **Time-limit expiry**, including any `OverTimeLimit` extension.
- **Launcher crash**, where the launching process dies before its children.

What the scheduler does

`scancel` without a signal sends SIGCONT, then SIGTERM, waits the configured KillWait, then SIGKILL. Cancelling a job step does not terminate the job (slurm.schedmd.com), so a step-only cancellation does not release the allocation, and signals other than SIGKILL are not sent to the batch step by default (slurm.schedmd.com). At the time limit each task is sent SIGTERM followed by SIGKILL (docs.coreweave.com), and an operator guide stresses that checkpointing or cleanup must finish within the KillWait window (docs.coreweave.com), and `OverTimeLimit` can extend a job beyond its limit by a number of minutes (man.archlinux.org). An optional `--signal` warning is not sent by default (manpages.debian.org), and its timing can vary because of the resolution of Slurm's event handling.

If a step does not die after SIGKILL, the unkillable-step logic applies: the timeout defaults to 60 seconds or five times MessageTimeout, whichever is greater, the node is drained when it is exceeded, and SchedMD support staff note that an `UnkillableStepProgram` can perform further actions when a step cannot be killed (support.schedmd.com). MessageTimeout itself defaults to 10 seconds (man.archlinux.org), which gives a 60 second floor on current defaults.

What the epilog does and does not prove

The epilog is a place to run a final check, not a guarantee. By default the Prolog runs on a node only when it first sees a step from a new allocation, whereas the Epilog runs on every node when the allocation is released (manpages.debian.org). `PrologFlags=RunInJob` runs both inside the extern `slurmstepd` so that they are contained in the job's cgroup (manpages.debian.org), which changes what device files they see. A practical epilog check is therefore read-only: list compute contexts on the node's GPUs by UUID, compare against the allocation, and exit non-zero on a mismatch so the node drains. This is a design suggestion, not a documented Slurm pattern, and it needs its own failure test so that a false positive cannot drain a healthy fleet.

Instruments for the observer

- **Cgroup state.** Read `cgroup.procs` and the `populated` key of `cgroup.events` before and after; use `cgroup.kill` only in a recovery step and record that it was used.
- **GPU contexts.** Query compute processes by GPU UUID. NVML's compute listing is a snapshot that reports only compute processes, and container PID namespaces make process lists unreliable because NVML is not compatible with them (nvidia.github.io); NVIDIA recommends monitoring on the host

(nvidia.github.io).

Separating slow cleanup from escape

Table 2 below defines the dispositions a harness should be able to emit. It separates the cases that look alike on an idle-utilization graph.

Observation	Disposition	Evidence required before assigning it
Job-owned GPU context present, process still in the job cgroup, within KillWait plus sampling interval	**Slow cleanup (expected)**	Timestamps showing the grace period has not elapsed
Process listed, state Z, GPU context absent	**Zombie, not a GPU holder**	Process state code and absence from the GPU compute list
Process in state D, survives SIGKILL past the unkillable-step timeout	**Unkillable step; drain**	State code, elapsed time versus timeout
Process holds a GPU context, `/proc/PID/cgroup` is outside the job cgroup, started via a systemd path	**Membership escape (fail)**	The cgroup path, ownership, and launch ancestry
Process with PPID 1 but still inside the job cgroup	**Reparented, not escaped**	`/proc/PID/cgroup` equals the job's path
Expected daemon context (for example a persistence daemon or monitoring agent)	**Expected context**	Process identity; the persistence daemon itself holds device files open
Case not run, route unsupported, or evidence not collected	**Unknown**	Written reason; counted in coverage as not executed

The key row is the fifth. A process with PPID 1 is not an escape by itself. Only the process's cgroup path, ownership and launch ancestry decide membership, and this table exists to record that evidence.

The disposition rules follow from the table:

- **Pass** only the routes and cases that were executed with complete evidence.
- **Fail** unexpected device access, or any retained job-owned GPU context once the grace period and sampling resolution are accounted for.
- **Unknown** applies to anything unsupported or unobserved.
- **Drain** nodes with unresolved results and escalate with minimal evidence (timestamps, cgroup paths, UUIDs, the configuration manifest).
- **Never** issue user-wide kills, and never infer isolation from idle utilization.

Implementation Considerations and Process Changes

Build the manifest first

Every result is only meaningful against a manifest, so record it before the first case. At minimum capture:

- **Slurm:** the version and the active `slurm.conf`, `cgroup.conf` and `gres.conf` as loaded by the daemons, kept consistent across nodes.

- **Kernel and systemd:** the kernel version, whether the unified cgroup v2 hierarchy is mounted, and the systemd version. Slurm states that it does not support a hybrid mode that mixes v1 and v2 controllers, so mixed mounts are a manifest failure before any case runs.
- **Plugin choice:** the effective cgroup plugin, read from the daemon log or configuration dump and not inferred from the file, because the documented default is autodetect.
- **Driver and launcher:** the NVIDIA driver version, and the launcher, image and runtime flags.
- **Access path:** the PAM and `sshd` configuration.

Resolve identifiers; do not compose paths

Because SLUID replaced the numeric job ID in v2 directory names from 26.05, harnesses must resolve the cgroup path from a process of the job and never build `job_<id>` strings. Use a three-step reconciliation:

- **Step 1:** read the GPU identities from inside the job (UUID and PCI bus ID).
- **Step 2:** read the scheduler's view of the allocation, remembering that `SLURM_JOB_GPUS` is not relative to any device cgroup.
- **Step 3:** map device-file minor numbers to UUIDs through `nvidia-smi`, whose output documents the minor number as the key to the `/dev/nvidia<minor>` node (docs.nvidia.com).

The device-file minor number is not the NVML index, so a script that works with paths needs this mapping. A mismatch in any step is recorded as a mapping finding and not as an isolation failure until the three agree.

Change SSH and PAM only with recovery prepared

The PAM stack is where a mistake locks administrators out or silently disables adoption. Treat it as a controlled change:

- **Retain recovery access.** Keep an existing root session or console open, because the module always allows root but a mis-ordered stack can still block everyone else. Do not copy a generic PAM stack from another site.
- **Respect ordering.** SchedMD states that the order of modules is very important (slurm.schedmd.com). A `required` module's failure is acted on only after the rest of the stack has run (manpages.ubuntu.com), and a failed `sufficient` module is ignored, so a bypass module placed ahead of `pam_slurm_adopt` changes its reach.
- **Know the defaults.** The default `action_no_jobs` is deny, `action_unknown` adopts into the newest job (raw.githubusercontent.com), and `action_adopt_failure` lets the connection through without adoption (slurm.schedmd.com). The generic-failure default falls through to the next module even if this one is marked required. SchedMD warns that plugin failures may otherwise let users log in unconfined; the adoption-failure test must deliberately cause a failure (for example by stopping `slurmd` on the canary) and record which path the session took.
- **Mind systemd.** Adopted sessions conflict with `pam_systemd`, which registers a session and creates a new systemd scope (man7.org); a SchedMD engineer states that systemd can steal cgroups back from Slurm unless it is disabled (support.schedmd.com), and logind can end the user's systemd instance when the last session closes (www.freedesktop.org).

- **Verify the SSH prerequisites.** `UsePAM` enables account and session module processing (manpages.ubuntu.com); `ForceCommand` can restrict compute-node login but does not replace the account check (man.openbsd.org); `pam_access` can add access rules (manpages.ubuntu.com). If adoption conditions are not met, SSH sessions persist even after the job ends (slurm.schedmd.com). The adoption check depends on the extern step created when `PrologFlags=Contain` is set, and jobs started before that flag was enabled cannot be adopted. Check the session closes: PAM sessions are terminated with a call to `pam_close_session` (man7.org).

Job containers

Slurm's `job_container/tmpfs` plugin removes files in its namespace when the job completes; on the page's description it addresses per-job filesystem namespaces, so this report credits it with temporary-file cleanup and not with device control. Slurm ignores a requested OCI container when `oci.conf` is absent on the execution node, so the test must confirm the container was applied.

Data Analysis and Evidence

No fault injection, isolation test, security audit or benchmark was performed for this report, and no numerical demand estimate for the target queries is available; the figures below are documented defaults and a labeled arithmetic example. *Table 3* below collects the documented defaults that bound cleanup timing and the failure policy, as published in the current SchedMD documentation as of November 2026.

Setting	Documented default or behavior	Why the acceptance test cares
KillWait	30 seconds between SIGTERM and SIGKILL (manpages.ubuntu.com)	Defines the grace period to subtract from cleanup latency
UnkillableStepTimeout	60 seconds or five times MessageTimeout, whichever is greater	Upper bound before a node drains for an unkillable step
MessageTimeout	10 seconds	Sets the 50 second term in the timeout formula (5 x 10)
OverTimeLimit	Zero minutes by default; minutes a job may exceed its limit	Delays time-limit termination when non-zero
UnkillableStepProgram	None runs by default	Without it, no automatic evidence is captured
ConstrainDevices	Default "no"	Device enforcement is absent unless set
PrologFlags	No flags by default (manpages.ubuntu.com)	The extern step needed for SSH adoption requires `Contain`
action_adopt_failure	Lets the connection through without adoption	An unconfined login is a possible default outcome
sbatch `--signal`	No signal before end time by default	No warning window unless requested

Cleanup latency and sampling resolution

Define **cleanup latency** as t(last confirmed job-owned GPU context released) minus t(termination requested). Because release is observed by polling, the observed value is an interval and not a point: if the last sample that showed the context was at time a and the first clean sample at time b, the release lies between a and b, and the latency is reported as the interval from a minus t(termination requested) to b minus t(termination requested). Grace periods are reported separately: the KillWait grace applies only if the process ignored SIGTERM.

(Hypothetical Example) Suppose `scancel` is issued at 14:00:00 and the harness samples GPU compute contexts every 5 seconds. The last sample showing the job's context is 14:00:41 and the first clean sample is 14:00:46. The cleanup latency is reported as 41 to 46 seconds. If the workload ignored SIGTERM, 30 seconds of that interval is the KillWait grace, leaving a residual of 11 to 16 seconds attributable to teardown; if the workload handled SIGTERM at once, the whole 41 to 46 seconds is teardown and deserves a closer look.

Tested-case coverage

Define **tested-case coverage** as executed applicable cases divided by declared applicable cases, always reported with failed and unknown counts. It is a statement about the test, not about security.

(Hypothetical Example) A site declares 24 applicable cases: 4 routes, each with a positive control, three negative controls and two termination cases. It executes 20: 18 pass and 2 fail, and 4 are not executed. Coverage is 20 / 24, or 83.3 percent, with 2 failed and 4 unknown. The node stays drained because failures and unknowns are open.

Existing references and community evidence

NVIDIA's AICR validator is the existing executable reference for one route: its Slinky probe records open errno values and UUIDs and decides pass or fail in separate evaluate functions, and it skips the Kubernetes per-pod probe for Slinky (docs.nvidia.com) because the node is reserved for the Slurm daemon pod.

Implications and Future Directions

- **Version drift is the main operational risk.** Slurm states that cgroup v1 receives no new features from 25.05 (slurm.schedmd.com) and marked it deprecated in the 25.05 release notes; the 26.11 notes add a `dmem` cgroup controller for shard GRES device memory (slurm.schedmd.com). A harness that encodes one cgroup layout will age quickly, so it should resolve paths and capabilities at run time and store the manifest with each result.
- **Cleanup evidence can be a standing control.** The same cases can run as a gate after image, launcher or kernel changes, and as a regression suite after a driver update, since the driver can rewrite device files whose attributes differ from its defaults (download.nvidia.com).
- **Open questions remain.** The current documentation reviewed does not state that an SSH-adopted process inherits the device constraint, so that route needs measured evidence at each site. MIG and AMD paths need their own mappings before they join the matrix.

- **Review.** Changes to authentication stacks and kernel device policy affect production access. Sites should have PAM and cgroup changes reviewed by their own security and platform engineers; no reviewer has examined this article.

Frequently Asked Questions (FAQs)

How does Slurm GPU cgroup isolation work, and what is `ConstrainDevices` in `cgroup.conf`?

`ConstrainDevices` constrains a job's allowed devices based on its GRES, using the devices subsystem, and defaults to "no". On cgroup v2 the devices controller is replaced by an eBPF program that `slurmstepd` creates and inserts. Only the devices described in `gres.conf` are managed.

Why can `srun` and `sbatch` show different GPU visibility?

A step inherits all the job's GRES by default but can be given none by setting the GRES count to zero. The visibility variable is set when tasks launch on the compute node, and the batch value and step values are separate artifacts to record in the matrix.

What cleans up a GPU after a job, and what does the epilog do?

Slurm's cancel sequence ends in SIGKILL after KillWait, and `proctrack/cgroup` is the recommended way to track every process. The epilog runs on every node when the allocation is released and drains the node if it fails. It is a place for a read-only GPU check, not a substitute for observing release.

Does `pam_slurm_adopt` restrict GPU access for SSH sessions?

It prevents SSH logins to nodes where the user has no job and tracks SSH-launched processes for cleanup (hpc-wiki.itc.rwth-aachen.de), but its default adoption-failure policy lets a connection through without adoption. Whether the adopted session is device-constrained must be measured with the device-open test.

Conclusion

A Slurm GPU allocation is only as isolated as the weakest of its layers, and each layer has its own evidence. Counting, masking, kernel-enforced device access, process membership and post-job release can each pass or fail independently, and the launch path (batch, `srun`, SSH adoption, or the site's container or serving launcher) determines which layers apply. The acceptance matrix in this report turns those separate controls into a written contract with positive and negative controls, stable identity reconciliation, timestamped termination cases, a failure taxonomy and a coverage figure that reports failed and unknown counts honestly.

The decision rule is deliberately conservative:

- **Pass** only the routes that were exercised and completely evidenced.
- **Fail** unexpected device access or retained job-owned GPU contexts.
- **Mark unknown** every unsupported or unobserved case, keep unresolved nodes drained, and escalate with minimal evidence.

- **Avoid** killing processes by user name and inferring isolation from idle utilization.
- **Re-run** the matrix whenever the image, launcher, PAM stack, driver or kernel changes, and read results together with the manifest they were produced against.

Source links

Links cited in this article, in order of first appearance. Full addresses are provided for readers using extracted text.

1. https://slurm.schedmd.com/gres.html
2. https://manpages.ubuntu.com/manpages/resolute/man5/gres.conf.5.html
3. https://man.archlinux.org/man/cgroup.conf.5.en
4. https://docs.nvidia.com/aicr/latest/contributor-guide/validators
5. https://raw.githubusercontent.com/SchedMD/slurm/slurm-25.05/RELEASE_NOTES.md
6. https://raw.githubusercontent.com/SchedMD/slurm/slurm-26.05/src/plugins/cgroup/v2/cgroup_v2.c
7. https://raw.githubusercontent.com/SchedMD/slurm/slurm-26.05/RELEASE_NOTES.md
8. https://slurm.schedmd.com/pam_slurm_adopt.html
9. https://manpages.debian.org/testing/slurm-client/scancel.1.en.html
10. https://man.archlinux.org/man/slurm.conf.5.en
11. https://docs.nvidia.com/datacenter/tesla/mig-user-guide/latest/device-nodes-and-capabilities.html
12. https://manpages.ubuntu.com/manpages/resolute/man5/slurm.conf.5.html
13. https://docs.nvidia.com/cuda/cuda-programming-guide/05-appendices/environment-variables.html
14. https://hpc.ncsu.edu/RunningJobs/Gpu.php
15. https://support.schedmd.com/show_bug.cgi?id=6412
16. https://slurm.schedmd.com/prolog_epilog.html
17. https://docs.nvidia.com/deploy/nvidia-smi/index.html
18. https://man.archlinux.org/man/gres.conf.5.en
19. https://slurm.schedmd.com/srun.html
20. https://slurm.schedmd.com/gres.conf.html
21. https://slurm.schedmd.com/cgroups.html
22. https://docs.kernel.org/admin-guide/cgroup-v1/devices.html
23. https://docs.kernel.org/admin-guide/cgroup-v2.html
24. https://slurm.schedmd.com/cgroup_v2.html
25. https://www.freedesktop.org/software/systemd/man/latest/systemd.resource-control.html
26. https://docs.ebpf.io/linux/program-type/BPF_PROG_TYPE_CGROUP_DEVICE
27. https://download.nvidia.com/XFree86/Linux-x86_64/570.169/README/faq.html
28. https://slurm.schedmd.com/cgroup.conf.html
29. https://man7.org/linux/man-pages/man7/cgroups.7.html
30. https://awslabs.github.io/ai-on-sagemaker-hyperpod/docs/slurm-orchestration/tips/enable-cgroups

31. https://man7.org/linux/man-pages/man2/PR_SET_CHILD_SUBREAPER.2const.html
32. https://man7.org/linux/man-pages/man2/setsid.2.html
33. https://man7.org/linux/man-pages/man2/PR_SET_PDEATHSIG.2const.html
34. https://man7.org/linux/man-pages/man2/wait.2.html
35. https://man7.org/linux/man-pages/man5/proc_pid_stat.5.html
36. https://man7.org/linux/man-pages/man7/signal.7.html
37. https://docs.nvidia.com/deploy/driver-persistence/latest/persistence-daemon.html
38. https://lwn.net/Articles/855049
39. https://manpages.debian.org/testing/slurm-client/srun.1.en.html
40. https://computing.ch.cam.ac.uk/slurm-usage
41. https://slurm.schedmd.com/containers.html
42. https://apptainer.org/docs/user/1.5/gpu.html
43. https://nvidia.github.io/cuda-python/cuda-bindings/latest/module/generated/cuda.bindings.nvml.device_get_compute_running_processes_v3.html
44. https://docs.coreweave.com/products/sunk/manage_sunk/manage_cluster_access/pam-slurm-adopt
45. https://hpc-wiki.itc.rwth-aachen.de/hpc/Admin_Guide_SSH_to_Compute_Nodes
46. https://docs.aws.amazon.com/en_us/parallelcluster/latest/ug/tutorials_11_running-containerized-jobs-with-pyxis.html
47. https://www.sherlock.stanford.edu/docs/software/containers/enroot
48. https://github.com/NVIDIA/enroot
49. https://slurm.schedmd.com/scancel.html
50. https://docs.coreweave.com/products/sunk/run_workloads/handle-slurm-signals
51. https://manpages.debian.org/testing/slurm-client/sbatch.1.en.html
52. https://support.schedmd.com/show_bug.cgi?id=11103
53. https://manpages.debian.org/testing/slurm-client/slurm.conf.5.en.html
54. https://nvidia.github.io/container-wiki/toolkit/faq.html
55. https://manpages.ubuntu.com/manpages/noble/man5/pam.d.5.html
56. https://raw.githubusercontent.com/SchedMD/slurm/slurm-26.05/contribs/pam_slurm_adopt/pam_slurm_adopt.c
57. https://man7.org/linux/man-pages/man8/pam_systemd.8.html
58. https://support.schedmd.com/show_bug.cgi?id=11255
59. https://www.freedesktop.org/software/systemd/man/latest/pam_systemd.html
60. https://manpages.ubuntu.com/manpages/noble/man5/sshd_config.5.html
61. https://man.openbsd.org/sshd_config
62. https://manpages.ubuntu.com/manpages/noble/man8/pam_access.8.html
63. https://man7.org/linux/man-pages/man3/pam_open_session.3.html
64. https://slurm.schedmd.com/release_notes.html

THE PUBLISHER

About GPU Smith

GPU Smith provides independent engineering and research for private AI infrastructure. We help technical buyers and operators reason about GPU workload sizing, hardware selection, procurement, deployment and inference operations, from the compute system to the networking, power and cooling requirements around it.

Start with the workload

Useful infrastructure decisions begin with the models, data, throughput, latency and operating constraints a team actually has. GPU Smith helps connect those requirements with system design and deployment choices. The goal is a privately controlled AI environment whose capacity and operational demands are understood before a purchasing decision.

Hardware and supplier research

Our public reference library covers GPUs, complete systems and networking components. The server-vendor directory and manufacturing research help teams investigate suppliers, compare options and follow sources behind technical claims. We distinguish manufacturer specifications from measured benchmarks and advertised capabilities from tested configurations. Reference pages are research resources, not a live inventory listing or a binding equipment quote.

Deployment and operations

GPU Smith's engineering scope includes infrastructure integration, networking, power, cooling and the practical operation of inference workloads. Our published articles explain the tradeoffs behind deployment, procurement and ongoing operations so teams can ask better questions and document decisions.

Work with GPU Smith

Explore the hardware library, GPU references, networking references and vendor research. Contact GPU Smith with your workloads and deployment constraints to discuss a project and confirm current scope, pricing and availability.

Inclusion of a manufacturer or product in our research does not imply a partnership, certification or endorsement.

Website: https://gpusmith.com

This article is educational. Verify consequential medical, legal, financial, regulatory and technical decisions with appropriate professionals and the cited primary sources. Company services are described separately from the article's research findings.